USE SENSE URGENCY TO CONTINUE WITH OTHER HEURISTICS TO DETERMINE SWITCH EVENTS IN A TEMPORAL MULTITHREADED CPU

BACKGROUND OF THE INVENTION

[0001] Temporal multithreading is known in the art as a technique that uses one set of execution resources to execute multiple "programs," or "threads." These execution resources often include an array of pipeline execution units. Instructions for a program thread are processed through the pipeline until it stalls; in a "switch" event, those stalled instructions are then removed and instructions from another thread are injected to the same pipeline so as to efficiently utilize execution resources.

[0002] Temporal multithreading thus gives the appearance of multiple central processing units ("CPU"). Each thread processes through the execution units as if the program had the entire control of the execution units; activation and deactivation of various threads occurs in hardware control logic based on multiple switching events in an attempt to maximally utilize the execution units.

[0003] There is a penalty associated with the above-mentioned switch events. Accordingly, the prior art has developed certain objective criteria for a switch event. In one example, a cache miss triggers a switch event because the processor needs to acquire data from main memory. In another example, a time out counter counts the cycles of a thread's execution and promotes an automatic switch for an out-of-bounds thread execution duration.

[0004] There is the need to further reduce the negative effects of switching events in high performing processors. By reducing or improving processing of switch events, a processor will have increased performance, by improving instruction processing efficiency across multiple threads. One feature of the invention is therefore to provide a processor with intelligent logic for efficiently processing and switching multi-threaded programs through the processor. Several objects and other features of the invention are apparent within the description that follows.

SUMMARY OF THE INVENTION

[0005] The following patents provide useful background to the invention and are incorporated herein by reference: U.S. Patent No. 6,188,633; U.S. Patent No. 6,105,123; U.S. Patent No. 5,857,104; U.S. Patent No. 5,809,275; U.S. Patent No. 5,778,219; U.S. Patent No. 5,761,490; U.S. Patent No. 5,721,865; and U.S. Patent No. 5,513,363.

[0006] In one aspect, a processing unit of the invention has multiple instruction pipelines for processing instructions from multiple threads. Though not required, each thread may include an urgency identifier within its program instructions. The processing unit has a thread switch controller to monitor processing of instructions through the various pipelines. The thread controller also controls switch events to move from one thread to another within the pipelines. The controller may modify the urgency of any thread such as through modification of the urgency identifier.

[0007] "Urgency," as used herein, generally means an abstraction of how well a thread is progressing (or will be progressing) within a pipeline; it is also an abstraction as to how urgent the program or processor logic believes the thread should be. By way of example, a thread's urgency may be "low," "medium" or "high," or further quantized with an 3-bit identifier, thereby providing different switch event solutions to treatment of the thread in stalled pipelines. If for example a thread instruction misses the cache, the controller can lower the thread's urgency, e.g., from high to medium, or from 5 to 4. Additional cache misses can lower the thread's urgency further, such as from medium to low, or from 4 to 3. The thread controller preferably utilizes certain heuristics in making switch event decisions. Other heuristics may be used by the controller to prescribe the switch events according to certain guidelines. By way of example, processor interrupts may also modify the switch event heuristics.

[0008] The controller also preferably monitors the timing of a thread, such as to incorporate time out features with the switch event heuristics.

[0009] The invention thus provides certain advantages. Unlike the prior art, switch events are no longer "black and white" decisions that may cause negative processing effects within the pipeline. By way of example, in accord with the invention, a switch event may not automatically occur after a certain number of cycles associated with time slice expiration. That is, an assessment is also made of a thread's execution and

relative to time slice expiration: if that thread is making good forward progress, it is not stalled or switched out; rather its use urgency may actually be increased to amplify the thread's activity. If however there is a stall, the next inactive thread might be activated. If on the other hand the active thread has a time slice expiration but has a higher urgency than the inactive thread, the inactive thread may remain dormant while the pipeline waits to process the active, more-urgent thread. Those skilled in the art should also appreciate that switching too late may also create processing penalties. The invention provides advantages over the prior art by adding urgency to the thread in order to encourage, or not, a switch event under appropriate heuristics.

[0010] Moreover, too many switch events may underutilize the pipeline. Once again, the invention has advantages over the prior art by reducing the underutilization of execution units, due to switch events, by making appropriate switch decisions according to preferred instruction processing policies; and these policies may be changed, dynamically or otherwise, to further reduce the underutilization. By way of example, a program thread with "low" urgency may switch immediately in the event of a stall (predicted or actual)

[0011] The invention is next described further in connection with preferred embodiments, and it will become apparent that various additions, subtractions, and modifications can be made by those skilled in the art without departing from the scope of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] A more complete understanding of the invention may be obtained by reference to the drawings, in which:

[0013] FIG. 1 schematically illustrates a processing unit of the invention for processing instructions through pipeline execution units;

[0014] FIG. 2 shows an exemplary switch event within a pipeline of the invention; and

[0015] FIG. 3 shows a flowchart illustrating the use of urgency with multithreading, in accord with the invention.

DETAILED DESCRIPTION OF THE DRAWINGS

[0016] FIG. 1 shows a processing unit 10 of the invention. Unit 10 is for example part of an EPIC processor to process multiple program threads through multiple pipelines 12. Pipelines 12 include an array of pipeline execution stages, known to those skilled in the art, to process instructions incrementally, such as in the fetch stage F, the register read stage R, the execute stage E, the detect exception stage D, and the write-back stage W. Unit 10 preferably has multiple instruction pointers 14(1), 14(2)... 14(M) to accommodate processing multiple threads 15(1), 15(2)... (15(M) through units 12. A cache 13 buffers data from associated pipelines 12 to register files 16. The plurality of register files 16(1), 16(2)... 16(N) provide per-cycle storage of data for unit 10, via bus 18; various architected states may be stored in register files 16(1), 16(2)... 16(N), including write-back data from the W stage. Bypass logic 20 may be used to accommodate bypass and speculative data transfers to and between pipelines 12 and register file 16.

[0017] Instructions are dispatched to pipelines 12 by a fetch unit 24 in communication with an instruction pointer 14. An issue unit 26 may be used to couple instructions into pipelines 12 for execution therethrough.

[0018] Unit 10 also includes a thread controller 30. Controller 30 monitors processing of threads within pipelines 12; it also defines the switch events that deactivate and activate multiple threads within pipelines 12 to perform multithreading. In the event of a switch event, instructions from current threads are routed through controller 30 via bus 17.

[0019] A time slice expiration unit 19 may also couple to unit 10 to monitor time slice expiration of any thread 15 within pipelines 12. Unit 19 couples with thread controller 30, via bus 21, to indicate an expiration event. In such an event, and as described below, thread controller 30 may switch out the current thread, or not, based on the urgency for that thread.

[0020] Thread controller 30 utilizes the urgency of various threads processed within unit 10 to define the switch events. By way of example, controller 30 may monitor a time slice expiration of a thread within pipelines 12 and elect to switch that thread out, or not, based on the urgency of the thread. Controller 30 may also modify the urgency of a thread, such as by injecting an instruction into the pipeline. For example, if a thread

repeatedly has cache misses, controller 30 may repeatedly lower the urgency of the thread, for example lowering the urgency bits of the thread's instructions from 3, to 2, to 1. In still another example, thread controller 30 monitors processor interrupts for a given thread; it may again adjust the thread's urgency based on the interrupts.

[0021] For purposes of illustration, FIG. 2 illustrates a switch event 70 defined by controller 30 within a pipeline 12. A first thread, illustratively with a "low" urgency, is executed (EX1) in a stage 60 of pipeline 12. That thread stalls at time 62 due to a cache miss. Controller 30 switches the first thread out, at switch event 70, and activates a second thread, illustratively with a "high" urgency.

[0022] The invention thus provides for executing instructions from other threads in the event a current thread stalls in the pipeline (e.g., EX1), such as through a cache miss that results in a long latency memory operation. More particularly, switch event 70 causes only a minor delay in pipeline 12. Stall 62 is covered by continued execution of another thread through the pipeline; specifically, the second thread continues execution (EX2) within pipeline 12 at stage 64. The second thread may also stall at time 66; however no switch event occurs, in this example, because the second thread has higher urgency than the first thread. Accordingly, stall 66 is covered by continued execution of the second thread within execution (EX2) within pipeline 12 at stage 68.

[0023] FIG. 3 shows a flowchart 100 illustrating certain non-limiting thread controller operations of the invention. After start, in step 102, the thread controller monitors processing of instructions within an array of pipelines. In step 103, the time slice expiration unit assesses whether a time slice expiration occurs. If no, processing continues to step 104. In the event a time slice expiration occurs, the active thread urgency is set to high and deactivated, and the inactive thread is activated, at step 105. Monitoring of another thread then starts anew, as shown.

[0024] In the event of a possible switch event, step 104, the thread controller assesses whether the event is for an active thread (step 106) or an inactive thread (step 108). If the event is associated with an active thread, the active thread urgency is modified and assessed against the inactive thread urgency to determine whether to switch, or not, at step 110. If the urgencies do not warrant a switch, monitoring continues at step

102. If the urgencies do warrant a switch, then a switch occurs at step 112. At step 112, the active thread is deactivated and the inactive thread is activated.

[0025] If the event from step 104 is for an inactive thread, step 108, then the inactive thread urgency is modified and assessed against the active thread urgency to determine whether to switch, or not, at step 114. If the urgencies do not warrant a switch, monitoring continues at step 102. As above, if the urgencies do warrant a switch, then a switch occurs at step 112. At step 112, the active thread is deactivated and the inactive thread is activated.

[0026] Accordingly, the invention assesses a thread's urgency to decide whether the current thread should be switched out of the pipeline or not (steps 110, 114). A modification of the thread's urgency may also occur by operation of thread controller 30, such as by issuing an instruction to "hint" of a new thread urgency.. If switched out, the next thread is activated within the pipeline. The controller then continues to monitor instruction progress of the new thread in the pipeline (step 102). Thread switches may occur by flushing the pipeline and switching to another architected state (e.g., via pointers to the register file and stored in the thread switch controller); the fetching of instructions from the new thread then commences. If the current thread is not switched out, for example if its urgency is very high, then it may not be switched out and the continued monitoring (step 102) occurs for the same thread.

[0027] The invention thus attains the objects set forth above, among those apparent from the preceding description. Since certain changes may be made in the above methods and systems without departing from the scope of the invention, it is intended that all matter contained in the above description or shown in the accompanying drawing be interpreted as illustrative and not in a limiting sense. It is also to be understood that the following claims are to cover all generic and specific features of the invention described herein, and all statements of the scope of the invention which, as a matter of language, might be said to fall there between.